Drools Syntax Diagrams

Wolfgang Laun

February 10, 2011

Preface

This is a comprehensive set of syntax diagrams for Drools 5.1. They were produced from an EBNF representation of the grammar using Peter Thiemann's program Ebnf2ps, available from `http://www.informatik.uni-freiburg.de/~thiemann/haskell/ebnf2ps/`.

Some language constructs were omitted because they are deprecated or just alternative forms, which may become deprecated as well.

The diagrams follow the usual conventions. Starting point is the first diagram, *CompilationUnit*. Each diagram has an entry point and an end point. You can follow possible paths between these two points by following the arrows through the boxes. A nonterminal, for which there is another diagram, is represented by a square box with yellow background while a terminal is represented by a round box and green background.

There are no diagrams for several nonterminals; these follow the syntax as defined by the Java Language Specification:

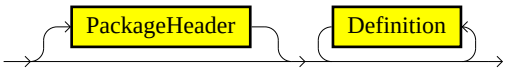
- *StringLiteral*
- *Identifier*
- *Statement*
- *Expression*

The DRL Parser Diagrams

The diagrams in this section describe the syntactic units of the DRL parser, which operates on tokens assembled by the lexer.

Compilation Unit

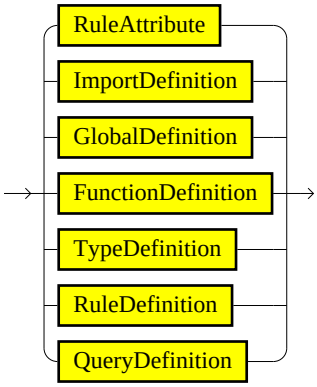
CompilationUnit



PackageHeader

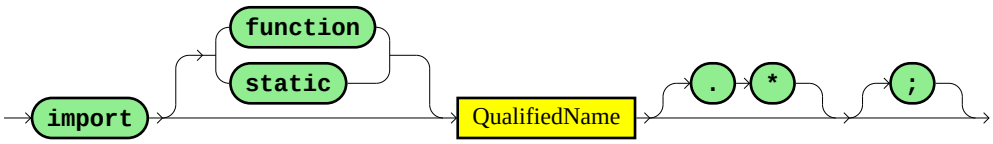

```
package com.acme.order;
```

Definition



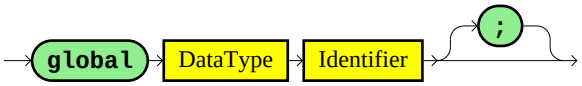
Import and Global

ImportDefinition




```
import com.acme.order.Position;
import com.acme.order.*;
import function com.acme.util.calculate;
import static java.lang.Math.PI;
```

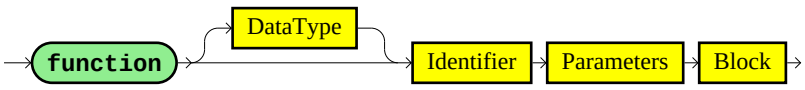
GlobalDefinition




```
global Map ident2product;
```

Function Definition

FunctionDefinition

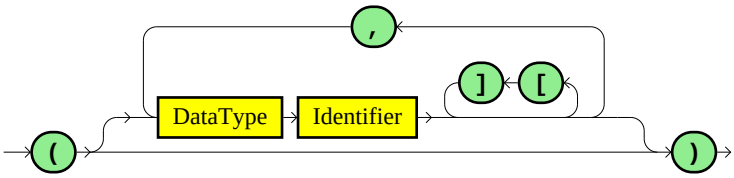



```
function sayHello(){
    System.out.println( "Hello!" );
```

```
}

function int sqr( int a ){
    return a*a;
}
```

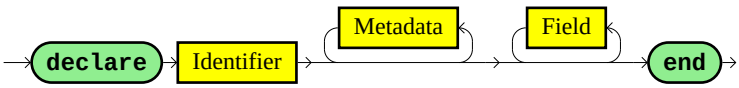
Parameters




```
function f( int i, String[] strings ){ /* ... */ }
```

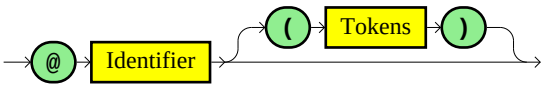
Type Definition

TypeDefinition



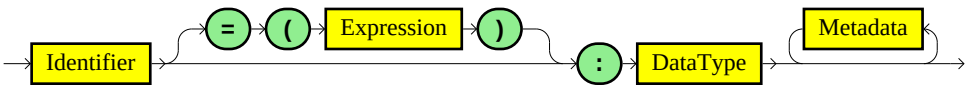

```
declare Message
    from  : Address;
    text  : String;
    state : MsgState;
end
```

Metadata




```
@author( John Doe )
@validated
```

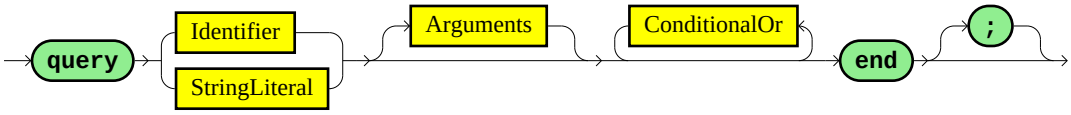
Field



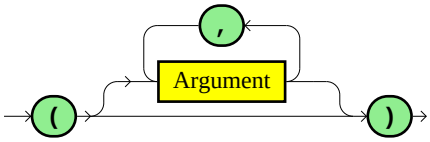
```
name : String
state = ( State.INITIAL ) : State @modifyBy( "rules" )
```

Query Definition

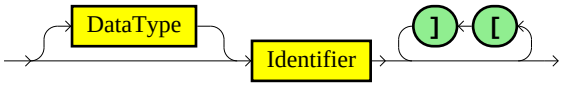
QueryDefinition



Arguments



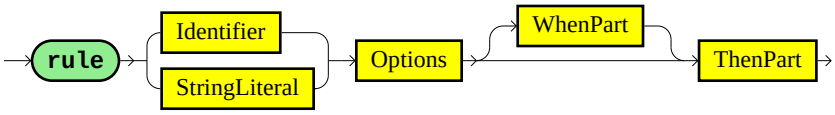
Argument



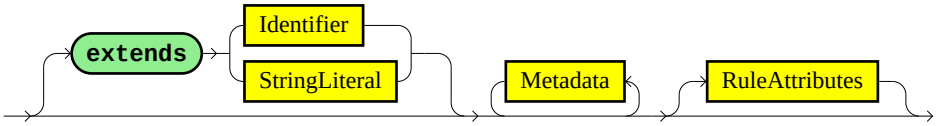

```
query ’any Item’
   $item : Item
end
query ’Triangles with side a’ ( Integer pa )
   $t : Triangle( a == pa )
end
```

Rule Definition

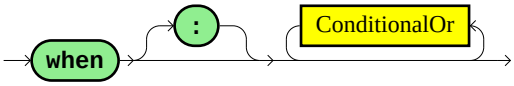
RuleDefinition



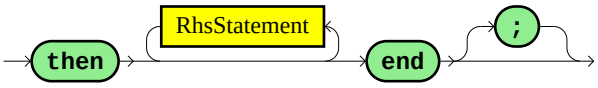
Options



WhenPart



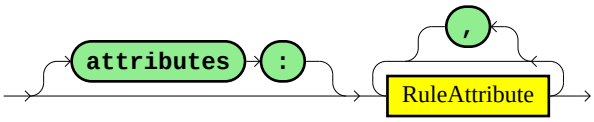
ThenPart



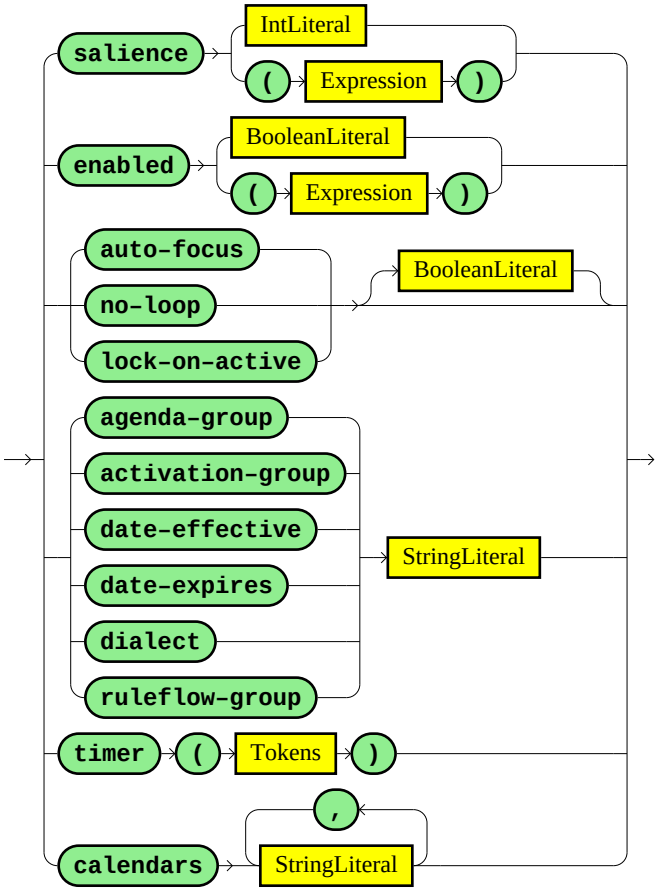

```
rule "concatenate paths"
when
    Path( $start1 : start, $goal1 : goal )
    Path( start == $goal1, $goal2 : goal )
then
    insert( new Path( $start1, $goal2 ) );
end
```

Rule Attributes

RuleAttributes



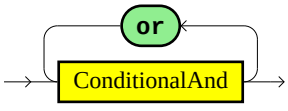
RuleAttribute



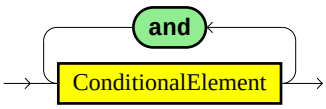

```
rule "assign Handler to Task"
    agenda_group( "assignHandler" )
    salience ( $prio )
when
    $t : Task( $prio : priority, $skill : skill, handler == null )
    $h : Handler( skills contains $skill, available == true )
then
    modify( $t ){ setHandler( $h ) }
    modify( $h ){  setAvailable( false ) }
end
```

Conditional Elements

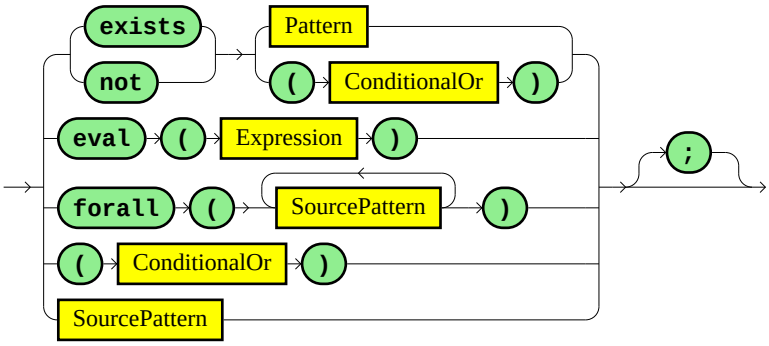
ConditionalOr



ConditionalAnd

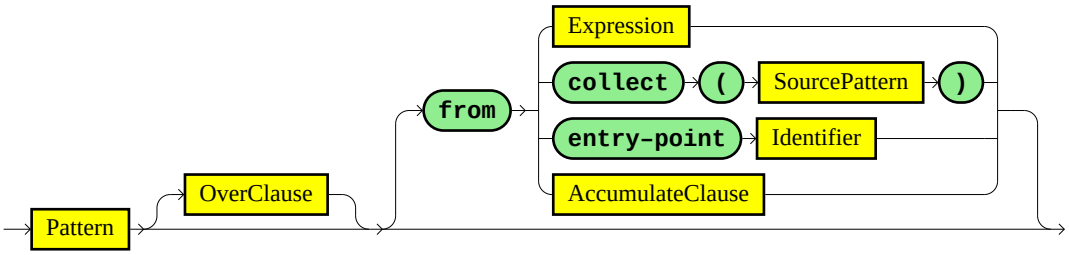


ConditionalElement

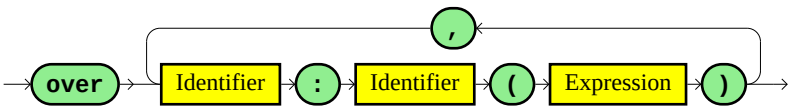



```
rule "CE demo"
when
    exists Fact( id == 1 )
    not ( Other( key == "foo" ) or Other( key == "bar" ) )
    Fact( id == 42, $data : data )
    eval( $data.size() > 10 )
then
    System.out.println( "Match!" );
end
```

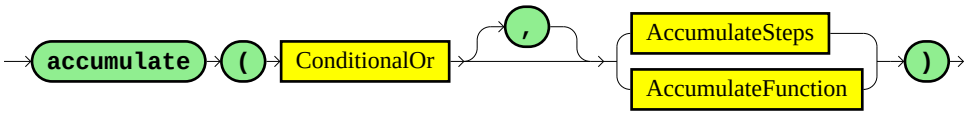
SourcePattern



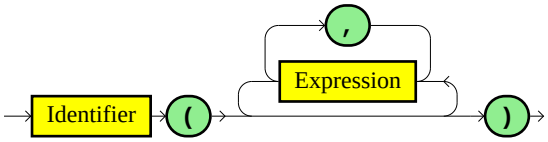
OverClause



AccumulateClause



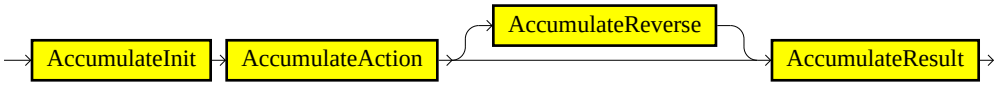
AccumulateFunction



Predifined accumulate functions are `sum`, `average` (or `avg`), `mininum` (or `min`), `maximum` (or `max`), `count`, `collectSet`, and `collectList`.

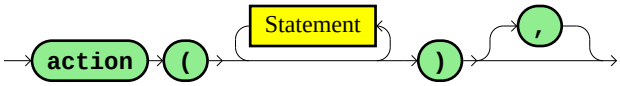
```
rule "Check item count"
when
    $batch : Batch( $id : id )
    $total : Number( intValue > 100 )
             from accumulate( Item( batch == $batch ), count() )
then
    System.out.println( "too many Items in Batch " + $id );
end
```

AccumulateSteps

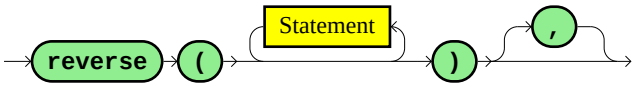


AccumulateInit

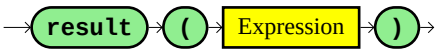
AccumulateAction



AccumulateReverse



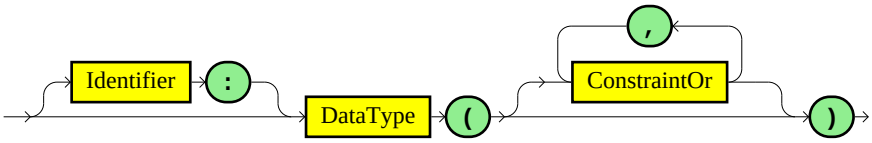
AccumulateResult




```
rule "print sorted Element names"
when
    $set: Set() from accumulate (
        Element( $name: name ),
        init( Set<String> nameSet =
            new TreeSet<String>( Collator.getInstance() ); )
        action( nameSet.add( $name ); )
        reverse( nameSet.remove( $name ); )
        result( nameSet ) )
then
    System.out.println( "Names: " + $set.toString() );
end
```

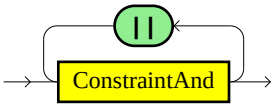
Pattern, Constraint and Restrictions

Pattern

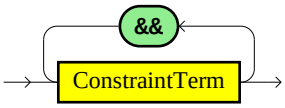



```
$emp : Employee( trainings contains "Java" )
```

ConstraintOr

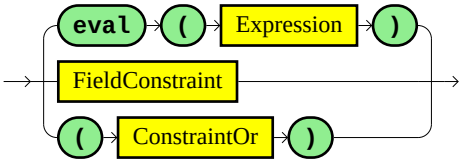


ConstraintAnd



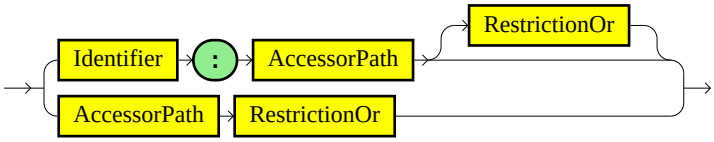

```
Resort( rate <= 100 || seaside == true && rate <= 110 )
```

ConstraintTerm



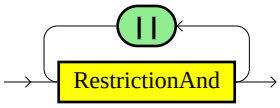
```
eval( $p1.getX() == $p2.getX() || $p2.getY() == $p2.getY() )
```

FieldConstraint

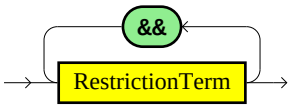



```
Wine( grape == "Merlot", $maker : maker )
```

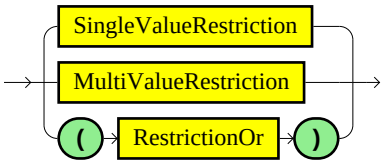
RestrictionOr



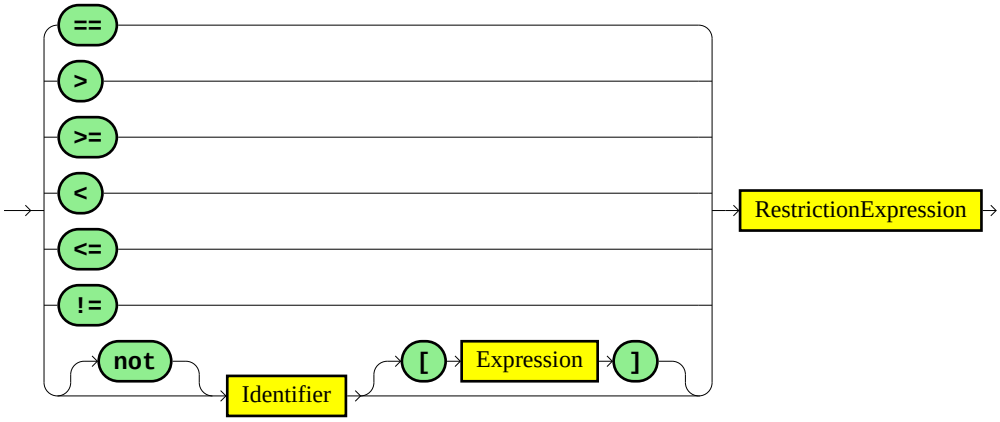
RestrictionAnd



RestrictionTerm



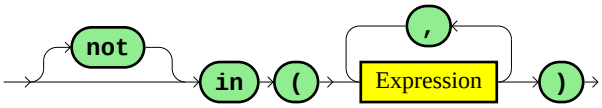
SingleValueRestriction



Predefined operators are: `contains`, `excludes`, `memberOf`, `matches`, `soundslike`, and the

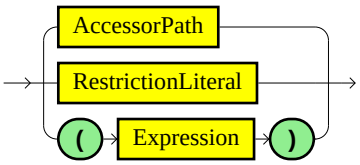
```
Medication( dose <= 100 || >= 200 )
```

MultiValueRestriction



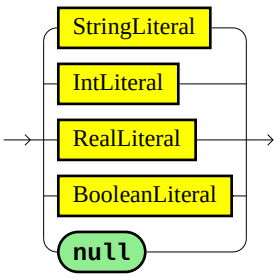

```
Duck( name in ( "Huey", "Dewey", "Louie" ) )
```

RestrictionExpression



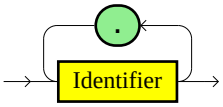

```
$t1 : Triangle()
$t2 : Triangle( this != $t1, a == $t1.a, b < 10, c < (2*a + b) )
```

RestrictionLiteral

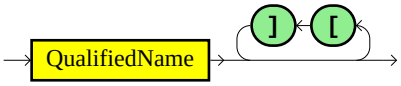


Miscellaneous

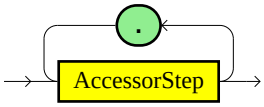
QualifiedName



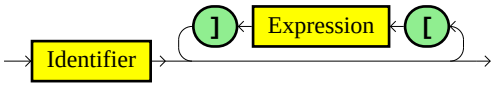
DataType



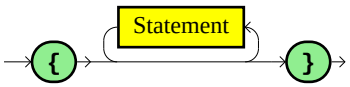
AccessorPath



AccessorStep

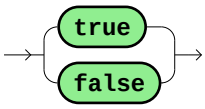


Block



BracketedExpression

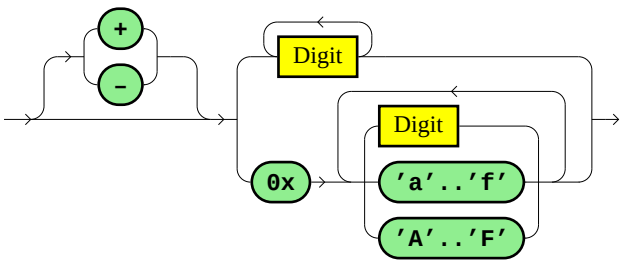
BooleanLiteral



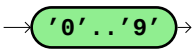
The DRL Lexer Diagrams

The diagrams in this section describe the syntactic units of the lexer, which assembles its non-terminals from individual characters. This means that its terminals may not be separated by any white space (space, tab, newline,...) from each other.

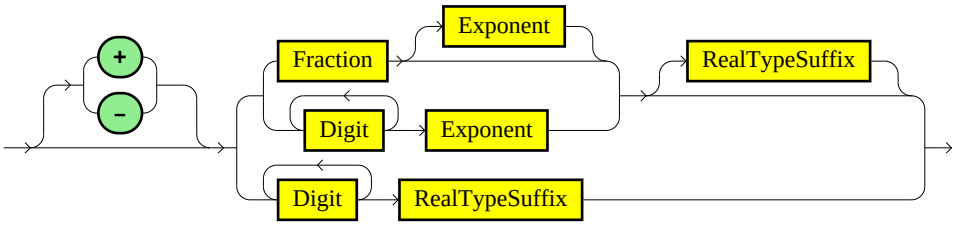
IntLiteral



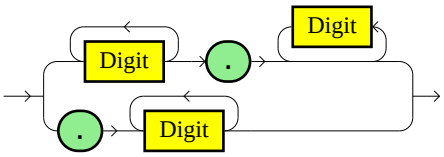
Digit



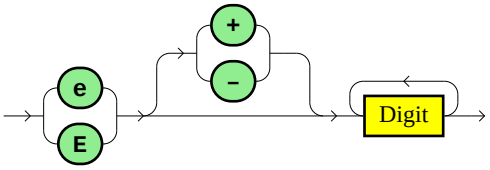
RealLiteral



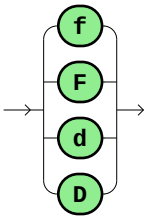
Fraction



Exponent



RealTypeSuffix

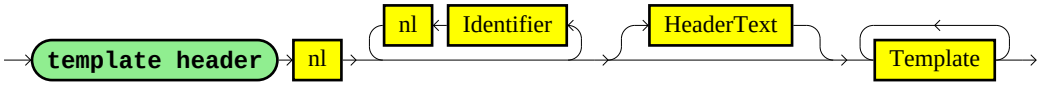



```
3.14159   .125   1e10   100f   2E-6D
```

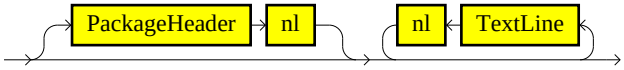
Template Files

The syntax of Template files is line-oriente; *nl* is used to indicate where a newline is required.

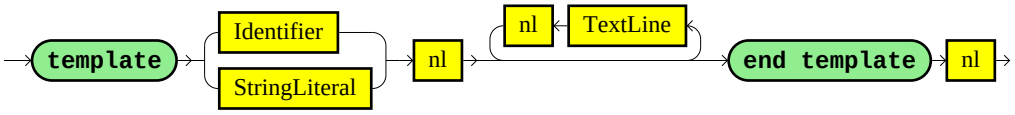
TemplateFile



HeaderText



Template




```
template header
type
test

template match
rule "match @{row.rowNumber}"
when
    f: @{type}( @{test} )
then
    System.out.println( "Match for " + f.toString() );
end
end template
```

Index